异构多核计算系统的 Codelet 任务调度策略 *

裴颂文[1,2]，吕春龙[1]，宁　钟[2]，顾春华[1]

(1. 上海理工大学 光电信息与计算机工程学院, 上海 200093; 2. 复旦大学 管理学院, 上海 200433)

摘　要：Codelet 数据流计算模型在处理大规模并行计算任务时效果显著，但该模型目前缺少在异构多核环境中的任务调度策略。因此，提出了一种在异构多核环境下基于蚁群算法的 Codelet 任务调度策略。该调度策略将启发式算法与蚁群算法相融合，在发挥各自优势的同时克服了启发式算法不能得出最优解的缺陷以及蚁群算法初始信息匮乏的问题。实验结果表明，智能蚁群任务调度策略相比 Codelet 运行时系统中原生的动态调度和静态调度策略具有更高的执行效率。

关键词：数据流计算；Codelet 模型；异构多核；蚁群算法；任务调度

中图分类号：TP183　　doi: 10.3969/j.issn.1001-3695.2017.12.0840

Codelet task sched uling policy of heterogeneous multicore computing system

Pei Songwen[1, 2], Lyu Chunlong[1], Ning Zhong[2], Gu Chunhua[1]

(1.*School of Optical-Electrical & Computer Engineering*, *University of Shanghai for Science & Technology*, *Shanghai* 200093, *China*; 2. *School of Management*, *Fudan University*, *Shanghai* 200433, *China*)

Abstract: Codelet dataflow model has significant effects on gaining high performance of computing large-scale parallel tasks, but the model currently lacks scheduling policy in heterogeneous multi-core environment. Regarding to this issue, this paper proposed a Codelet task scheduling strategy by fusing ant colony algorithm with a heuristic approach in heterogeneous multicore environment. It had both advantages of the heuristic algorithm and ant colony algorithm, and it overcomed both defects of the heuristic algorithm that could not derive an optimal solution and the defects of the ant colony algorithm that was lack of the initial information. The experimental result shows, the smart ant colony scheduling policy is much more efficient than the native dynamic and static scheduling policies in the runtime system implementation of the Codelet model.

Key words: dataflow computation; Codelet model; heterogeneous multi-core; ant colony algorithm; task schedule

0　引言

数据流计算模型是由麻省理工学院的 Dennis[1]提出，被认为是冯诺依曼控制流计算机在并行化计算方向上的突破性贡献。数据流计算模型执行指令的方式与传统的冯氏计算机基于指令控制流的方法不同，它采用有向图描述数据流模型，指令所需的数据一旦就绪即可触发该执行，具有高度的并行性。数据流计算模型充分发掘了程序内在的数据并行性，使用细粒度的并行机制克服传统控制流模型的局限性。为了提高多核系统的执行效率并解决数据流模型细粒度所引起的问题，混合数据流计算模型应运而生。混合数据流模型[2,3]结合了数据流和冯诺依曼控制流各自的优势，最大化利用硬件资源的同时，还能降低系统功耗。Codelet 模型[4,5]就是一种重要的混合数据流计算模型。

多核处理器已经成为主流计算机的重要计算部件，改变了以往处理器仅支持少量计算单元的应用环境。随着并行执行模型的不断完善, Suettlerlein [6]提出了 Codelet 模型，并基于此模型，提出了对应的运行时系统 DARTS。Codelet 模型是一种细粒度的、由事件驱动的、混合控制流/数据流的并行执行模型[7]。Codelet 模型由基本执行单位 Codelet 和线程程序 TP（threaded procedure）组成。其中，Codelet 主要负责存储执行指令，TP 作为 Codelet 的容器，负责输入、输出及保存共享数据。DARTS 作为 Codelet 模型的运行时系统，负责多核、众核上的任务调度、负载平衡、内存管理和功耗管理等。本文使用 DARTS 实现和模拟任务在 Codelet 模型中的执行过程。

多核处理器又分为同构多核和异构多核两种类型。Kumar 等人[8]指出，同构多核存在一定的局限性，如果在芯片上集成

收稿日期：2017-12-18；**修回日期：**2018-02-09　**基金项目：**上海市自然科学基金资助项目（15ZR1428600）；上海市浦江人才项目（16PJ1407600）；中国博士后科学基金资助项目（2017M610230）；国家自然科学基金重点资助项目（61332009）；国家自然科学基金面上项目（61775139）

作者简介：裴颂文(1981-)，男，湖南邵东人，副教授，博士，硕导，主要研究方向为异构计算机系统结构、多媒体大数据分析、分布式计算(swpei@usst.edu.cn)；吕春龙（1992-），男，硕士研究生，主要研究方向为深度学习方法；宁钟（1964-），男，教授，博导，主要研究方向为管理调度、供应链管理、创新与创业；顾春华（1970-），男，教授，博导，主要研究方向为分布式计算系统、云计算等.

多个简单的内核，将会造成能耗和芯片冷却的负担。而异构多核处理器除了配置通用的处理器核外，还集成了一些特定功能的处理器核心，能够加快程序的执行速度并降低系统功耗[9]。异构多核处理器系统的发展受到广泛的关注并将成为未来处理器的发展方向，然而 Codelet 数据流计算模型缺乏在异构计算环境下的任务调度方案。

执行模型的选择是并行系统设计中最根本的问题，而调度策略[10~12]又决定了任务在执行模型的执行效率[5]。为了减少并行任务的执行时间，本文提出了一种面向异构 Codelet 的智能蚁群任务调度策略。通过改进蚁群算法，使其适用于异构 Codelet 模型，将各 Codelet 分配至指定的执行单元，从而完成任务在模型中的调度，最后使用标准任务图集中的有向无环图模拟仿真实验过程并与 DARTS 中自带的两种调度策略进行对比，以验证新调度策略的有效性。

1 Codelet 模型与蚁群算法

CodeletCodelet 是 Codelet 数据流计算模型的基本执行单元，由一系列机器指令组成，这些机器指令可看做是单一的、非抢占式的计算单元。在 Codelet 程序执行模型中，Codelet 是最基本的调度单位，任务被划分成许多相连通的 Codelet。与传统任务的并行机制不同，Codelet 的执行语义是当且仅当所有需要的资源可用时，Codelet 才会被激活。然后，Codelet 将根据调度策略分配给对应的目标处理单元，并且多个激活的 Codelet 可以并行执行。

1.1 DARTS

DARTS (delaware runtime system) [6]是实现 Codelet 数据流计算模型的运行时系统，其主要作用是根据调度策略分配硬件资源来加载和执行 Codelet。DARTS 运行时由四种对象类型组成：线程程序调度器(TPS)、Codelet 调度器(CDS)、抽象机配置和出口 Codelet。其中，TPS 主要负责线程程序和 Codelet 的负载均衡，并在空闲时执行已激活的 Codelet；CDS 的任务主要是执行对应执行单元上已激活的 Codelet；抽象机配置是运行时与用户交互的接口，可由用户设置 Codelet 模型的硬件环境；出口 Codelet 是任务在 Codelet 模型所执行的最后一个 Codelet，该 Codelet 亦即任务执行完毕的信号。本文仅考虑单一集群下的 Codelet 调度策略，在 DARTS 中，一个集群下有一个 TPS 和多个 CDS，调度器的数量一般等于或小于实际处理器所拥有的处理单元。

1.2 调度模型

在 Codelet 数据流计算模型中，任务调度的目标是寻找可在机器上运行的最优或近似最优的调度方案。Codelet 在依赖关系的约束下，尽可能地缩短任务执行时间。假定 m 是某个计算任务中 Codelet 节点的数量，n 是某个集群/多核处理器中计算单元（对应调度器）的数量，调度任务即是将 m 个 Codelet 分配给 n 个调度器。每个 Codelet 的计算都会独占调度器直到该 Codelet 的计算任务执行完毕。

本文采用 CDG（Codelet graph）模拟真实机器执行数据流计算任务时的 Codelet 调度场景。如图 1 所示，CDG 由 Codelet、token 和 event 三部分组成。Codelet 是基本的执行单位，token 表示 Codelet 执行完毕后发出的信号，event 表示两个 Codelet 间的依赖关系。CDG 可以明确地表示出 Codelet 的状态与 Codelet 之间的依赖关系。

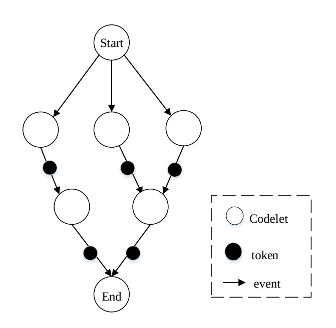


图 1 CDG（Codelet graph）

CDG 采用有向无环图 G=<V, E>描述 Codelet 数据流计算模型。Codelet 的节点构成顶点集合 V={ n_i }， $i \in$ [0,N-1],N 是 Codelet 任务数；边集 E={ $e_{i,j}$ }， $i, j \in$ [0，N-1]， $e_{i,j}$ 表示 Codelet 任务 n_i 与 n_j 之间的数据依赖关系。如表 1 所示，本文定义了 Codelet 模型中的数据结构和调度相关的参数。

表 1 符号和参数定义

符号	定义
$prec(i)$	n_i 的直接前驱集合
$succ(i)$	n_i 的直接后继集合
$C(i, j)$	n_i 和 n_j 之间数据传输时间
$allow(t)$	解的构造过程中，调度步骤 t 的就绪任务集合，包括所有前驱均已接受调度的 Codelet 任务
$FT(i)$	Codelet 任务 n_i 的结束时刻
$ETC(i, j)$	任务 n_i 在调度器 p_j 上的执行时间
$MK(s)$	s 的调度长度; $MK(s)=\max_{i\in[0,N-1]}(FT(i))$;异构 CDG 调度的优化目标为： $\min(MK(s))$
s	Codelet 节点的调度序列（调度方案）

1.3 蚁群算法

蚁群算法（ant colony optimization Algorithm，ACO）[13,14]最早由意大利学者 Marco Dorigo 等人在 20 世纪 90 年代提出，是一种从群体智能演化而来的解决大规模组合优化问题的算法。蚂蚁通过信息素传递信息，如果某个路径较短，则单位时间内通过的蚂蚁数量相对多，在该路径上残留的信息素就越多，后续蚂蚁将有倾向地选择该路径通行。蚁群算法正是利用了这一原理，通过这种积极的反馈，使其具有强大的全局趋同性。蚁群算法以信息素为载体，具有发现较优解的能力，可进行大范围的全局搜索，同时具备一定的鲁棒性和潜在的并行性。蚁群算法已经在组合优化、函数优化、网络路由、机器人路径规划、数据挖掘等领域获得了广泛的应用，并取得了较好的效果。本

文对蚁群算法进行改进，使其适用于异构环境下的 Codelet 模型的任务调度。

2　智能蚁群任务调度策略

本文面向异构多核的 Codelet 计算环境，融合启发式算法 HEFT(heterogeneous earliest-finish-time) [15]和蚁群算法提出了一种智能蚁群任务调度策略 SACTS（smart ant colony task scheduling）。HEFT 算法会在每个步骤中选择相距出口节点最远的 Codelet 计算节点，并将选择的 Codelet 分配给异构计算单元执行。智能蚁群任务调度策略会根据 HEFT 算法计算出每个 Codelet 计算任务的静态优先级，优先级越高，相应的 Codelet 就需要越早被调度执行。智能蚁群任务调度策略将各个 Codelet 的静态优先级引入到蚁群算法中作为调度 Codelet 计算任务的重要依据。

图 2 展示了本文提出的智能蚁群任务调度策略的流程。智能蚁群任务调度策略的具体执行步骤如下：①首先导入 CDG 图、数据传输的时间开销 C 以及任务执行开销 ETC；②根据 HEFT 算法计算出每个 Codelet 节点的静态优先级 $rank$；③蚂蚁从 Codelet n_{start} 开始节点出发，根据动态状态转移规则选择下一个调度的 Codelet 节点；④将选择的 Codelet 节点分配给能最早执行完成的调度器（计算单元）；⑤判断该蚂蚁是否遍历完所有的 Codelet 节点，如果是，则进行局部信息素更新，否则继续选择下一个 Codelet 节点；⑥判断所有蚂蚁是否都完成一次遍历，如果都完成遍历，则进行全局信息素更新，否则继续分配蚂蚁进行下一轮遍历；⑦记录本次迭代计算的最优解并更新动态因子；⑧判断是否达到最大迭代次数，如果是则输出最优解，否则继续下一轮迭代。

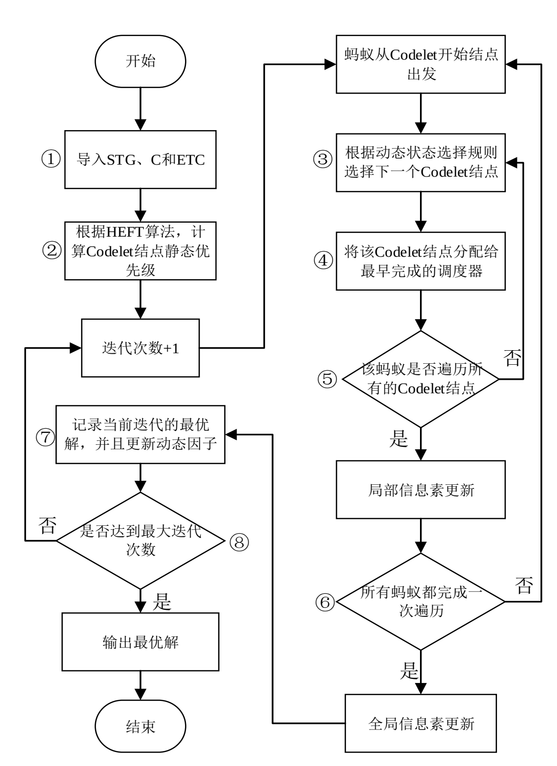


图 2　智能蚁群任务调度策略流程

2.1　参数定义

如表 2 所示，本文定义了智能蚁群任务调度策略所使用的参数和初始化值。

表 2　参数和初始化值

参数	参数的含义	初始化值
m	蚂蚁数量	50
n	迭代次数	200
τ_0	信息素矩阵初始值	0.001
β	启发式信息在蚂蚁选择路径中所起作用大小	1.2
q_0	动态因子，调节已知信息利用和新信息探索之间的比例	0.1
ρ	局部信息素挥发因子	0.1
φ	全局信息素挥发因子	0.1

2.2　构造调度序列的解

智能蚁群任务调度策略将蚁群算法的某个解 s 表示为 $s=\{n_0, n_1, \cdots, n_i, \cdots, n_{N-1}\}$。按照蚂蚁走过的路径，依次将 Codelet 任务节点添加到 s 队列中，s 队列满，则表示蚂蚁完成了一次遍历，蚂蚁移动的路径即 Codelet 的某个调度序列。智能蚁群任务调度策略执行过程中依赖 s 队列和 $allow$ 集合。s 队列依次记录了所有已经被调度的 Codelet 任务节点；初始化时所有的节点均未被调度，s 队列为空。$allow(t)$ 表示在选择第 t 个 Codelet 任务节点时，可供选择的 Codelet 任务节点集合；初始化时，$allow(0)$ 仅有开始节点 Codelet n_{start}。　如算法 1 所示，构造了智能蚁群任务调度策略的解。该算法循环 N 次，每次会选择一个 Codelet 任务节点记录在 s 队列中，然后将该 Codelet 任务节点分配给最早执行完成的调度器并且记录其执行完成的时间。当 s 队列满时，输出所有 Codelet 任务节点中最大的结束时间，即解 s 的调度长度（调度开销），解 s 即 Codelet 节点的调度序列。

算法 1　解的构造

输入：CDG 图 G

输出：$MK(s)$

1. for each ant k
2. build a queue s of ant path;
3. build a queue $allow$ of Codelets to be chosen;
4. t=0;
5. add Codelet n_{start} to $allow(t)$;
6. while t < N do

//通过动态转移规则从 $allow(t)$ 中选择下一个调度的 Codelet

7. choose Codelet n_i from $allow(t)$ by dynamic rule;

//选择能最早执行完成 Codelet 的调度器（计算单元）

8. choose a scheduler to execute Codelet n_i;
9. record FT of Codelet n_i;
10. add the Codelet n_i to s;
11. t=t+1;
12. update $allow(t)$;

13. end while

14. $MK(s)=\max_{i\in[0,N-1]}(FT(i))$;

15.end for

2.3 启发式信息定义

在智能蚁群任务调度策略中，η 为蚂蚁的调度操作提供启发式知识，能促使蚁群更快发现较优解或最优解，减少求解的盲目性。智能蚁群任务调度策略将 HEFT 算法中使用的启发式知识带入 η。在 HEFT 算法中，距离出口节点最远的任务最重要，应该先接受调度，因此 η 可以定义为 Codelet 节点的静态优先级 $rank(n_i)$，根据式(1)计算。

$$rank(n_i)=w_i+\max_{n_j\in succ(n_i)}(c_{i,j}+rank(n_j)) \quad (1)$$

其中：$succ(n_i)$ 是 Codelet n_i 的直接后继集合；$c_{i,j}$ 是边 (i,j) 的通信开销；w_i 是 Codelet n_i 在调度器上的平均计算开销。静态优先级 $rank$ 值是通过向上遍历 Codelet 任务图递归求得。对于 Codelet n_{exit}，其 $rank$ 值根据式(2)求得。

$$rank(n_{exit})=w_{exit} \quad (2)$$

智能蚁群任务调度策略在初始化时计算出每个 Codelet 的 $rank$，之后该值作为所有蚂蚁共有的信息，$rank$ 值越高，蚂蚁选择该 Codelet 的概率越高。

2.4 动态状态转移规则

蚂蚁通过状态转移规则决定下一个调度的 Codelet 任务。智能蚁群任务调度策略使用动态伪随机概率规则进行状态转移，见式(3)。

$$n_i=\arg\max_{w\in allow(i)}\{[\tau(i,w)][\eta(i,k)]^{\beta}\},\quad \text{if}\ \ q<q_0 \quad (3)$$

$\tau(i,w)$ 为 (i,w) 边上的信息素强度，$\tau(i,w)$ 值越大，蚂蚁选择 Codelet n_k 的概率就越大。$\eta(i,w)$ 即为 $\eta(n_w)$ 根据式(1)(2)计算可得。

在本文提出的动态伪随机概率规则中，通过调整动态因子 q_0 来控制收敛速度。q_0 值越小，按照式(4)随机性选择的概率就越大，蚁群选择的多样性会越好；反之，q_0 值越大，蚂蚁选择 $allow(i)$ 集合中 $[\tau(i,j)][\eta(i,j)]^{\beta}$ 值最高的 Codelet n_i 的概率就越大，收敛速度会加快。因此在智能蚁群任务调度策略初期设定较小的 q_0 值增加蚁群选择的多样性，而在后期随着迭代次数的增长将 q_0 值调整为较大的值可以加快收敛速度。根据式(4) q_0 值计算可得。

$$q_0=q_{\min}+\frac{(q_{\max}-q_{\min})*l}{n} \quad (4)$$

其中：l 表示当前迭代次数；n 表示最大迭代次数。

根据式(3)，蚂蚁在每次的状态转移时都会生成一个[0,1]之间的随机数 q。若 $q<q_0$，蚂蚁选择 $allow(i)$ 集合中 $[\tau(i,j)][\eta(i,j)]^{\beta}$ 值最高的 Codelet n_i；反之，蚂蚁根据式(5)计算 $allow(i)$ 中每个 Codelet 任务被选择的概率，进行随机概率选择。q_0 表示已知信息利用和新信息探索之间的比例。

$$p_i(j)=\begin{cases}\dfrac{[\tau(i,j)][\eta(i,j)]^{\beta}}{\sum_{w\in allow(i)}[\tau(i,w)][\eta(i,w)]^{\beta}}, & if\ w\in allow(i)\\ 0, & otherwise\end{cases} \quad (5)$$

2.5 信息素更新

在构造解的过程中，每一只蚂蚁通过局部更新规则对自己选择的边进行局部信息素更新。在一次迭代中，当所有的蚂蚁都完成了解的构造后，再对最佳路径上的边进行全局信息素更新。

2.5.1 局部信息素更新规则

局部信息素更新的目的是防止本轮迭代中的后续蚂蚁得到相同的解，因此在构造解过程中，蚂蚁挥发当前调度对应的信息素 $\tau(t,n)$，更新公式为

$$\tau(t,n)=\tau(t,n)*(1-\rho)+\tau_0*\rho \quad (6)$$

2.5.2 全局信息素更新规则

在一次迭代中所有蚂蚁完成解的构造后，对信息素矩阵进行全局更新，增加蚁群找到的最优调度步骤上的信息素，更新公式为

$$\tau(t,n)=\tau(t,n)*(1-\varphi)+\Delta\tau(t,n)*\varphi \quad (7)$$

$$\Delta\tau(t,n)=\frac{1+\max(0,MK(old)-MK(new))}{\min(MK(old),MK(new))},(t,n)\in gbbetter \quad (8)$$

式(8)中 $MK(old)$ 是本次迭代开始前算法得到的最优解，$MK(new)$ 是本次迭代找到的最优解，$gbbetter$ 是其中的更优者。

3 实验与分析

3.1 实验环境

实验环境配置为 Intel Core i7-6700HQ CPU，Nvidia GTX1080Ti GPU, 16 GB DDR4 主存，每个核包含 128 KB 的一级数据缓存和指令缓存，1 MB 的二级缓存和 6 MB 的三级缓存，CentOS-7-x86_64 操作系统。使用的运行时系统是基于 Codelet 模型设计的 DARTS。为了评价智能蚁群任务调度策略的性能，实验采用标准任务图集 STG（standard task graph）。STG[16]是一种评估多处理器调度算法的基准程序。本文在 STG 图结构不变的基础上，随机生成各节点在不同调度器上的执行时间以及 Codelet 节点在调度器之间的数据通信时间。

3.2 实验数据及结果分析

实验使用了包含 300、500、750、1 000、1 250、1 500 个节点的随机 STG 以及基于实际应用生成的 STG：Robot Control(88 个节点)、Sparse Matrix Solver(96 个节点)、SPEC fpppp(334 个节点)。在异构多核调度器环境中，比较了 DARTS 中两种适用于 Codelet 模型的调度策略（(静态策略(static policy)、动态策略(dynamic policy)[6]）和智能蚁群任务调度策略的任务执行时间。其中，静态策略采用轮询法将已激活的 Codelet 分配给指定的调度器。动态策略将所有可执行的 Codelet 存放在一个队列中，每当有调度器空闲时，都可单独访问该队列并获取 Codelet，因此动态策略适用于非对称的应用程序。

表 3 所示是三种调度策略执行包含不同节点的随机 STG 所需的时间。SACTS Policy 的执行时间相比于 Static Policy[6]减少了 34.7%~47.6%，SACTS Policy 的执行时间相比于 Dynamic Policy[6]减少了 5.7%~23.5%。

表 3　随机 STG 的执行时间

Codelet 数量	Static Policy[6]	Dynamic Policy[6]	SACTS Policy
300	148.8	103.0	97.2
500	251.5	174.2	152.3
750	377.4	264.1	225.7
1000	668.2	468.1	382.6
1250	853.0	584.6	447.2
1500	1200.1	781.5	635.8

图 3 所示是智能蚁群任务调度策略 SACTS 相比于静态策略和动态策略的执行时间加速比。加速比的计算公式为：某个调度策略执行任务时间/SACTS 执行任务时间。由图可知，SACTS 相比 Static Policy 和 Dynamic Policy 的执行时间加速比均大于 1，这说明 SACTS 要优于 DARTS 运行时系统原生的 Static Policy 和 Dynamic Policy。其中 SACTS 相比 Static Policy 的执行时间加速比在 1.53~1.90，相对于 Dynamic Policy 的执行时间加速比在 1.06~1.31。随着 Codelet 任务节点数量的增加，智能蚁群任务调度策略相对于静态策略和动态策略的执行时间加速比总体呈现递增状态，这说明随着 Codelet 任务节点规模的扩大时，智能蚁群任务调度策略的执行效率也在不断提高。

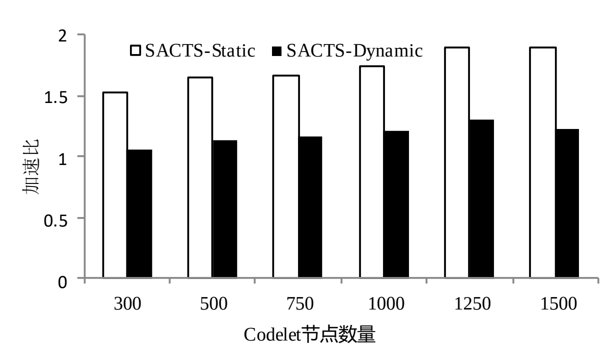


图 3　执行时间加速比

图 4 所示是三种调度策略执行基于实际应用生成的 STG 的执行时间。对于 robot，使用 SACTS 的执行时间比 Static Policy 减少了 24.7 s，比 Dynamic Policy 减少了 7.1 s；对于 sparse，使用 SACTS 的执行时间比 Static Policy 减少了 27.3 s，比 Dynamic Policy 减少了 8.4 s；对于 fpppp，使用 SACTS 的执行时间比 Static Policy 减少了 67.7 s，比 Dynamic Policy 减少了 20.5 s。实验结果表明，智能蚁群任务调度策略在处理实际应用时，也具有相当好的执行效率。

如表 4 所示，HEFT 以及 SACTS 在异构多核的 Codelet 计算环境下，对不同 STG 进行调度的执行时间。其中，SACTS(第一轮迭代)是指蚁群在迭代 1 次后得到最优调度序列，SACTS 是指蚁群在迭代 200 次后得到的最优调度序列。Codelet 任务在调度器的执行时间为[0,100]区间的随机变量，调度器之间的数据传输时间为[0,10]区间的随机变量。对于 Codelet 数量为 88(robot)、96(sparse)、334(fpppp)、300、500、750、1 000、1 250、1 500 的 STG，SACTS(第一轮迭代)比 HEFT 的执行时间减少了 10%~23.8%，SACTS 比 HEFT 的执行时间减少了 11.2%~26.3%。SACTS 执行时间比 HEFT 少的主要原因是蚁群在每一次迭代过程中都会增加最优路径上的信息素，通过这种积极的正反馈可以得到较优解或最优解。SACTS(第一轮迭代)即蚁群在迭代 1 次后得到最优调度序列的执行时间少于 HEFT 主要原因是 HEFT 算法中的启发式知识可以促使蚁群更快发现较优解或最优解，减少了求解的盲目性。实验结果表明智能蚁群任务调度策略的性能优于 HEFT，并且改善了蚁群算法初始信息匮乏的缺点。

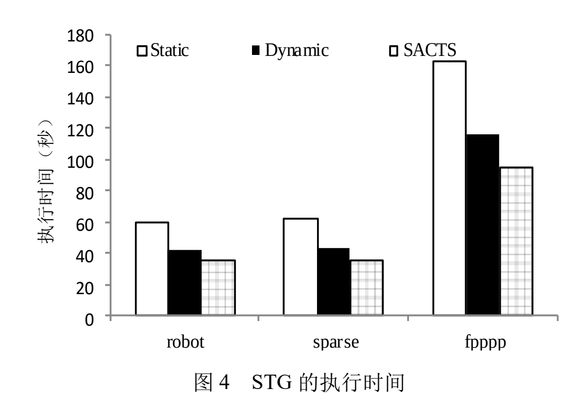


图 4　STG 的执行时间

表 4　STG 的执行时间

Codelet 数量	HEFT	SACTS(第一轮迭代)	SACTS　Policy
88(robot)	1993	1683	1557
96(sparse)	1870	1651	1584
334(fpppp)	6499	5263	5238
300	7117	5342	5239
500	11588	8834	8704
750	17208	14196	14018
1000	22454	19273	19027
1250	24585	22133	21826
1500	31915	27573	27240

4　结束语

随着 Codelet 数据流模型以及异构多核系统的发展，Codelet 任务调度策略成为制约异构 Codelet 数据流模型执行效率的重要因素之一。本文在异构多核环境下，研究了 Codelet 数据流模型的调度策略，并提出了智能蚁群任务调度策略。智能蚁群任务调度策略将启发式算法 HEFT[12]和改进的蚁群算法相融合，提高了模型的执行效率，缩短了任务执行时间。通过实验证明，在解决大规模 Codelet 任务调度问题时，智能蚁群任务调度策略与 Codelet 运行时系统中的原生动态调度和静态调度策略相比具有更高的效率。随着性能和速度的提高，处理器

的能耗和散热问题在设计高性能系统时成为关键的挑战。在 Codelet 数据流模型中，如何有效的降低能耗以及均衡处理器核心上的能耗，防止处理器上出现局部过热，是未来工作的研究重点。

参考文献：

[1] Dennis J B. First version of a data flow procedure language [C]// Proc of Symposium Programming. Berlin: Springer 1974: 362-376.

[2] Yazdanpanah F, Alvarez-Martinez C, Jimenez-Gonzalez D, *et al.* Hybrid dataflow//von-neumann architectures [J]. IEEE Trans on Parallel and Distributed Systems, 2014, 25 (6): 1489-1509.

[3] Grafe V G, Hoch J E, Davidson G S. Eps'88: combining the best features of von neumann and dataflow computing [R]. Albuquerque: Sandia National Labs, 1989.

[4] Zuckerman S, Suetterlein J, Knauerhase R, *et al.* Using a codelet program execution model for exascale machines: position paper [C]// Proc of the 1st International Workshop on Adaptive Self-Tuning Computing Systems for the Exaflop Era. 2011: 64-69.

[5] Suettlerlein J, Zuckerman S, Gao G R. An implementation of the Codelet model [M]// Wolf F, Mohr B, Mey D. Euro-Par 2013 Parallel Processing Workshops. Berlin: Springer. 2013: 633-644.

[6] Suetterlein J. DARTS: a runtime based on the Codelet execution model [D]. State of Delaware: University of Delaware, 2014.

[7] Pei S, Wang J, Cui W, *et al.* Codelet scheduling by genetic algorithm [C]// Proc of IEEE Trustcom//BigDataSE//ISPA. 2016: 1492-1499.

[8] Kumar R, Tullsen D M, Jouppi N P, *et al.* Heterogeneous chip multiprocessors [J]. IEEE Computer, 2005, 38 (11): 32-38.

[9] Augonnet C, Thibault S, Namyst R, *et al.* StarPU: a unified platform for task scheduling on heterogeneous multicore architectures [J]. Concurrency and Computation: Practice and Experience, 2011, 23 (2): 187-198.

[10] 赵国亮，李云飞，王川. 异构多核系统任务调度算法研究 [J]. 计算机工程与设计, 2014, 35 (9): 3099-3106.

[11] 裴颂文，宁静，张俊格. CPU-GPU 异构多核系统的动态任务调度算法 [J]. 计算机应用研究, 2016, 33 (11): 3315-3319.

[12] Nguyen T D, Vaswani R, Zahorjan J. Parallel application characterization for multiprocessor scheduling policy design [C]// Feitelson D G, Rudolph L. Job Scheduling Strategies for Parallel Processing. Berlin: Springer, 1996: 175-199.

[13] Chiang C W, Huang Y Q, Wang W Y. Ant colony optimization with parameter adaptation for multi-mode resource-constrained project scheduling [J]. Journal of Intelligent and Fuzzy Systems, 2008, 19 (4): 345-358.

[14] Chiang C W, Lee Y C, Lee C N, *et al.* Ant colony optimisation for task matching and scheduling [J]. IEE Proceedings-Computers and Digital Techniques, 2006, 153 (6): 373-380.

[15] Li M, Wang H, Li P. Tasks mapping in multi-core based system: hybrid ACO&GA approach [C]// Proc of the 5th International Conference on ASIC. 2003: 355-340.

[16] Tobita T, Kasahara H. A standard task graph set for fair evaluation of multiprocessor scheduling algorithms [J]. Journal of Scheduling, 2002, 5 (5): 379-394.